CS697 Thesis Research I – Final Report

Enhancing Security in RC Based P2P Networks: A Blockchain-Based Access Control Framework Utilizing Ethereum Smart Contracts

Submitted by:

Saurav Ghosh

Master of Science in Applied Computer Science

Advisor:

Dr. Reshmi Mitra

Assistant Professor

Department of Computer Science, Harrison College of Business & Computing

Southeast Missouri State University

Abstract:

Ensuring security for highly dynamic peer-to-peer (P2P) networks, critical for services like online transactions and smart devices, has always been a significant challenge. Adding to this dynamism and very high churn rates is their uniqueness in access control compared to traditional systems, particularly Role-Based Access Control (RBAC). This thesis contributes toward a new access control framework for blockchain-based systems using Ethereum smart contracts. The framework contributes toward closing the gap in existing systems with flexible, transparent, and decentralized security solutions. The contract framework we propose has access control contracts (ACCs) that protect access through static and dynamic policies, a Judge Contract (JC), and a Register Contract (RC) that records ACCs and JC for the purpose of their interactions. Its security model incorporates impact and severity-based assessment of the threats by using the CIA (Confidentiality, Integrity, Availability) and STRIDE principles, whose result will be responses tailored to the threats. The system does not only mitigate the fundamental volatility of peer membership but provides a scalable solution, which could be of high value, notably in areas like the Internet of Things (IoT) and Web 3.0 technologies.

1. Introduction

The introduction of peer-to-peer (P2P) networks has been very essential to decentralized service distribution, including services such as online transactions and content sharing or interconnectivity of intelligent devices. However, despite their great usefulness, P2P networks face the whole scale of fundamental security challenges that are topped up by the high churn of members and their structural openness. Usually, these challenges of the system come in the following form of threats under the STRIDE model: Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service, Elevation of Privilege—all serious risks to the confidentiality, integrity, and availability.

Traditional access control models, such as Role-Based Access Control (RBAC), have insufficiencies in managing the dynamism and decentralization that characterize P2P networks, and hence, give rise to serious trust issues among the parties of a network. Most of the time, these trust issues arise mostly due to a lack of centralized authority. This would ensure that there is an existence of some form of an authority that ensures all participants are authenticated to perform tasks within the network and given permission to do so in the right manner, and that their actions within the network are accountable.

In this work, we introduce a blockchain-based framework for access control to address these challenges by using the natural characteristics of blockchain technology, which comprise immutability, transparency, and decentralization. Our RC-based architecture uses a two-level hierarchy, which can ensure efficient data lookup and, consequently, minimizes latency in communication. At level 1, a group head (Gh_i) is connected with an optimization for data routing structure. At level 2, there are fully connected groups of peers, where each group (Gi) is attached to the transit network through its group head (Gh_i). In a fully interconnected network, therefore, groups communicate between themselves and never to the transit network. The architecture proposes the access control through the powerful Ethereum Smart Contract framework and integration of it into the framework makes it a trustless environment, so it can enforce and acknowledge valid security policies that do not require central authority intervention and, therefore, mitigate existing possible security threats.

1.1 Motivation

The motivation of this research is the current gaps in traditional methods that offer a scalable and adaptive way for the access control to state-of-the-art (SoA) consensus security mechanisms within P2P networks. This proposed system, based on blockchain, aims at bridging these gaps with further enhancement of the overall infrastructure of security, allowing much stronger and reliable network interactions.

1.2 Problem Statement

Peer-to-peer (P2P) networks face significant security challenges due to their decentralized nature and high member turnover. Very often, traditional control systems like RBAC (Role-Based Access Control) very often leave security loopholes wide open for unauthorized access and possible data breaches, precisely because they fail to be applied in a network management scenario that is dynamic. In essence, this research will cater to a critical need for a flexible, scalable trustless security framework in P2P networks, whose aim is to mitigate a range of dynamic security threats for enhanced decentralized operational integrity, while at the same time ensuring sophisticated communication among the network architecture, access control, Web 3.0 ethos, and the strengths of blockchain. It can be positioned in the transformational landscape of the distributed infrastructure and web 3.0, which indicates cybersecurity challenges.

1.3 Research Questions

1. How can blockchain technology be used to address the inherent security vulnerabilities in peer-to-peer (P2P) networks, particularly regarding access control?
2. In what ways do Ethereum smart contracts be used in the implementation of access control mechanisms in P2P networks compared to traditional access control models?
3. How does the proposed blockchain-based access control framework ensure dynamic and static policy management within P2P networks, and what are its effects for network security?

2. Literature Review:

2.1 Related Research on Peer-to-Peer Security in General

The need for robust security frameworks in IoT environments is critical. Novo (2018) proposed scalable blockchain architecture for the access management of IoT, presenting the nature of blockchain that can provide better security for the systems and bring about efficacy in their operations. Going further than that, Yu, Chen, and Wang (2022) discuss other layers of security challenges, like those represented by latency and node heterogeneity problems, including in distributed networks. Now, respectively, Mbarek, Ge, and Pitner (2020) apply those same concepts to smart home, smart grid systems, showing how data integrity and resilience of blockchain towards cyber-attacks significantly increase for these special applications within peer-to-peer networks. [1][3][12][18].

2.2 Blockchain in Peer-to-Peer Networks

Sun et al. (2022) critically reviewed the role that blockchain plays in IoT, with a great focus on how it can revolutionize access control across different sectors by using smart contracts with improved transparency. Complementing this, Ding et al. (2019) provide an attribute-based access control scheme minimizing overhead, enhancing security, and proofing real-world practicability. Similarly, Wang et al. (2022) proved their development —a dynamic access control system that will adapt to the changes in user behavior and the status of his devices in real-time—stressing the flexibility of blockchain solutions under a complex network environment [2][16][19].

2.3 Access Control in Peer-to-Peer Networks

On the other hand, another scope of Liu et al. (2020) related to the integration of blockchain in IoT networks is its great impact on enhancing access control mechanisms, thus reducing the risks brought by unauthorized accesses and breaches. Furthermore, Sultana et al. (2020), Badhe and Arjunwadkar (2023) provided the most insightful practical examples with regard to these themes: detailed and analyzed practical examples of how blockchain-based smart contracts find application, enabling secure and efficient access control systems for data sharing and data distribution. It draws an apt example of scalability and adaptability, important for contemporary IoT frameworks [8][11][13][15][17].

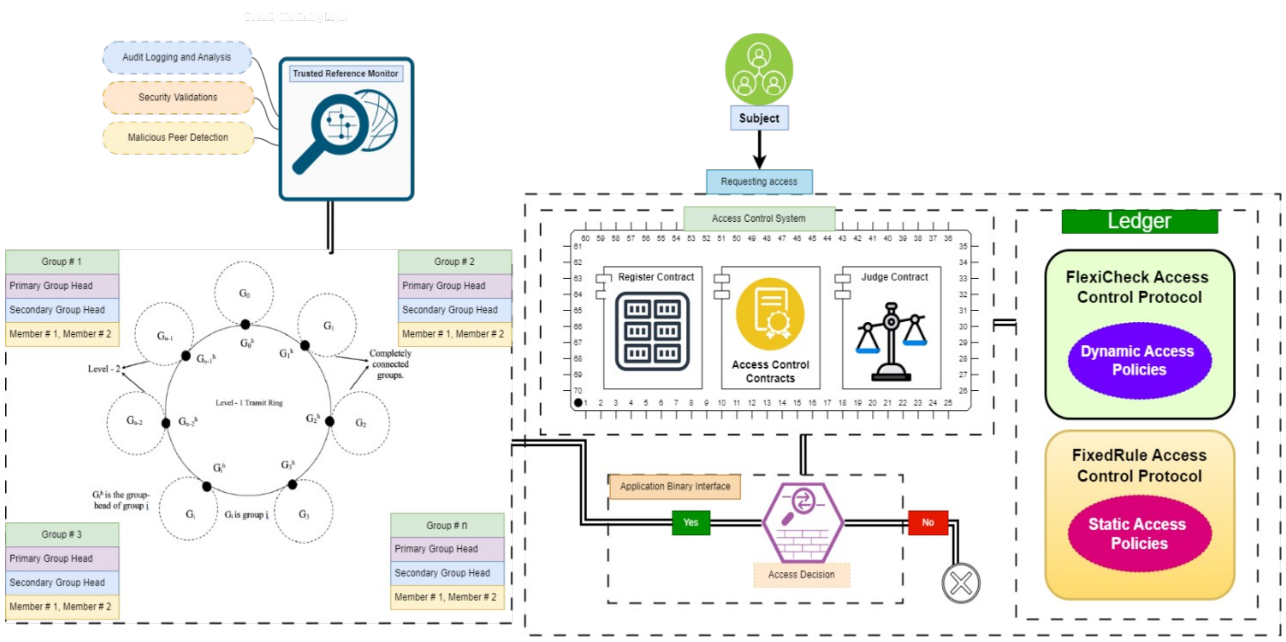
2.4 Integration of Access Control, Peer-to-Peer, and Blockchain

Collaboration with the access control system through the blockchain model to peer-to-peer networks is a strong solution against IoT security. Zhang et al. (2020) have elaborated a critical evaluation of the different blockchain-based models and scope on the key conditions of improvement for the scalability and security concerns. Han et al. (2022) and Li et al. (2021) further elaborated with an audit dual-layer scrutinization of the blockchain system to possibly manage access controls more efficiently, which consequently enhances a better reality of reliability and flexibility in operations. Such developments would be critical in moving forward towards more sophisticated and robust IoT architectural designs that can handle the dynamic and complex nature of modern network interactions [5][6][7][9][10][14][20]

3. Methods

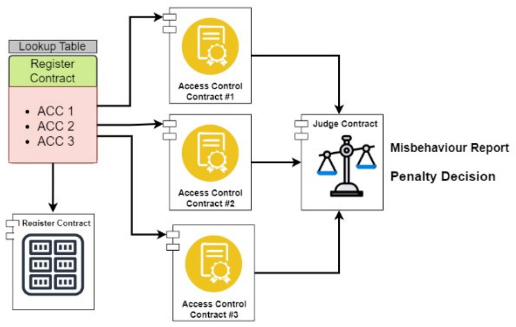
3.1 Design Philosophy

Access Control Process Flow The process flow for access control in the proposed system is as follows:



3.2 Smart Contract System

3.2.1 Access Control Architecture

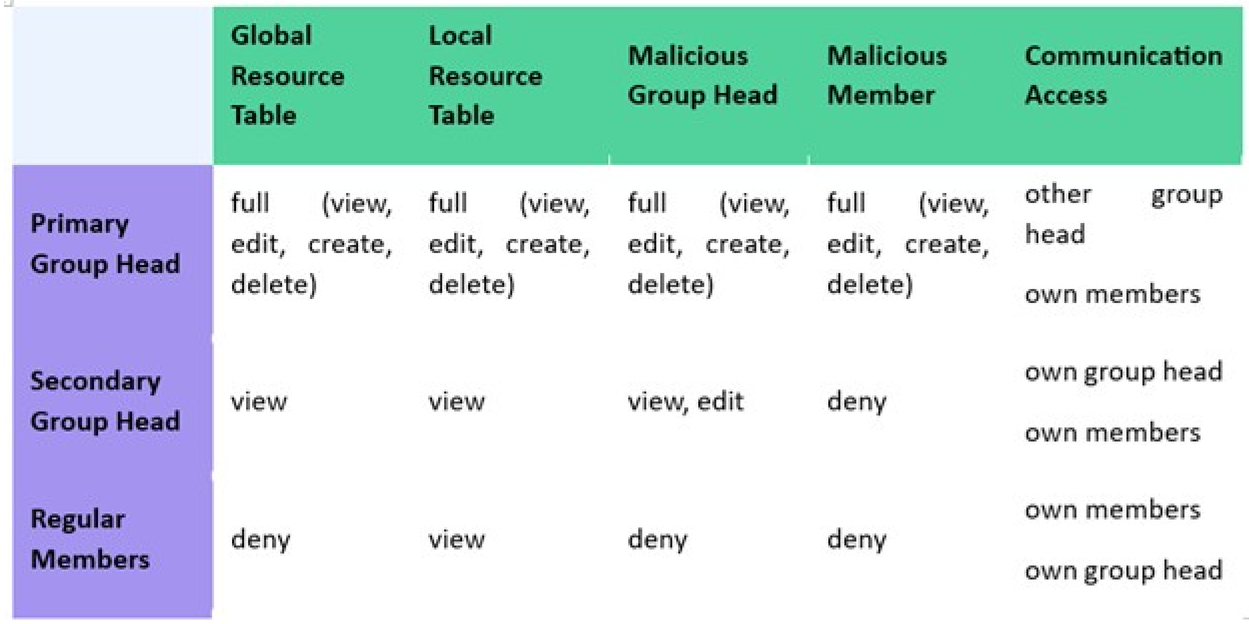


The proposed framework consists of multiple access control contracts (ACCs), and each implements the access control for a pair of peers a.k.a. one subject and one object. And there are one judge contract (JC), which takes the misbehavior report of a peer from an ACC, judges the misbehavior and determines the corresponding penalty, and one register contract (RC), which stores the information of the JC and ACCs and provide functions to manage these contracts and as well, stores every log event data to help the JC impose penalty.

3.2.2 Access Control Contracts (ACCs)

At the core of the framework, designed to secure peer-to-peer (P2P) networks, are the Access Control Contracts (ACCs), as illustrated in Figure 3. These contracts, such as ACC 1, ACC 2, and ACC 3, are deployed by a peer (the object) who seeks to regulate access requests from another peer (the subject). It's an approach where the control of access is managed between the subject and object peers and each ACC is exclusively dedicated to one specific pair, ensuring a tailored and secure access control mechanism.

Each ACC operates on a dual validation system, where necessary, to manage access requests from the subject peer effectively. The first layer, Fixedrule access control protocol, where static access right validation, inspects access requests against a set of predefined policies to ensure they comply with established permissions. The dynamic layer of validation takes this a step further by observing the subject's behavior in real-time. This allows the system to adaptively manage access based on current activities, enhancing security against potential threats or abuse.



	Global Resource Table	Local Resource Table	Malicious Group Head	Malicious Member	Communication Access
Primary Group Head	full (view, edit, create, delete)	full (view, edit, create, delete)	full (view, edit, create, delete)	full (view, edit, create, delete)	other group head own members
Secondary Group Head	view	view	view, edit	deny	own group head own members
Regular Members	deny	view	deny	deny	own members own group head

An illustrative example of how an ACC functions can be understood through its management of a policy list. This list acts as the backbone of the ACC, where each entry delineates a policy for a specific resource-action pair. Here's a glimpse into what such a policy list might look like, adapted to reflect a more comprehensive and role-specific scenario:

Role	Resource	Action	Permission	ToLR
Primary Group Head	Global Resource Table	Edit	allow	2023-04-01 10:00:00
Primary Group Head	Local Resource Table	delete	allow	2023-04-01 10:00:00
Regular Members	Communication Access	with Own Group Head	allow	2023-04-01 10:00:00
Regular Members	Local Resource Table	delete	deny	2023-04-01 10:00:00
Secondary Group Head	Global Resource Table	edit	deny	2023-04-01 10:00:00
Secondary Group Head	Local Resource Table	view	allow	2023-04-01 10:00:00

This example list log data for various actions on resources, defined by the role of the peer. Overall, this setup serves as the foundation for static validation by outlining whether a specific action on a resource is allowed or denied. The "Time of Last Request (ToLR)" field adds a dynamic aspect to the validation process, enabling the ACC to monitor and control access based on the recency of requests, thereby preventing potential misuse through too frequent access requests in a short timeframe.

Furthermore, the ACC maintains a misbehavior list for each resource. This list records any deviations from acceptable behavior by the subject, the time of occurrence, and the penalties imposed. Such meticulous record-keeping strengthens the framework's ability to dynamically adjust access permissions and enforce discipline within the P2P network.

3.2.2.1 Dynamic Validation Policies with Flexicheck Access Control Protocol

This table presents a range of security policies designed to protect IoT systems by dynamically responding to different types of security violations.

Security Event	Severity	CIA Impact	STRIDE Impact	Response
Too many Access Attempts	High	Integrity	Elevation of Privilege	24-hour ban
Data Tampering	High	Integrity	Tampering	Cancel access privileges permanently
Unauthorized Access to	Medium	Confidentiality	Information Disclosure	Temporary suspension of

Security Event	Severity	CIA Impact	STRIDE Impact	Response
Restricted Operations				access rights for 7 days
Disruption of Service	High	Availability	Denial of Service	30 days suspension, possible legal action
Misrepresentation of Identity	Low	Accountability	Spoofing	Warning for first offense, the penalties for repeat offenses

To manage these policies and implement the access control, the ACC offers several Application Binary Interfaces (ABIs), including but not limited to, **policyAdd**, **policyUpdate**, **policyDelete**, and **accessControl**. These functions enable the ACC to add new policies, update or delete existing ones, and execute both static and dynamic validations for access requests.

• *policyAdd()*: This ABI receives the information of a new access control policy and adds the information to the policy list.

• *policyUpdate()*: This ABI receives the information of a policy that needs to be updated and updates thepolicy.

• *policyDelete()*: This ABI receives the identification information of a policy and deletes the policy.

• *accessControl()*: This ABI receives the information required for access control and returns the access resultand penalty.

• *setJC()*: To send the report to JC, ACC needs to have this ABI.

3.2.3 Judge contract (JC)

The JC judges the misbehavior of the subject and deter-mines the corresponding penalty, when receiving a potential misbehavior report from an ACC. After determining the penalty, the JC returns the decision to the ACC for further operation.

• *Object*: the peer who felt the misbehavior.

• *Misbehavior*: the details of the misbehavior.

• *Time*: the time when the misbehavior is exhibited; and

• *Penalty*: the penalty imposed on the misbehavior.

Subject	Object	Misbehavior	Penalty
Primary Group Head 1	Local Resource Table	Unauthorized edit	Blocked 2 hours
Primary Group Head 2	Global Resource Table	Excessive deletion	Blocked 1 day
Secondary Group Head	Malicious Member	False report	Blocked 3 hours
Primary Group Head 3	Secondary Group Head's member	Unauthorized access	Warning issued
Secondary Group Head	Primary Group Head's resource	Data tampering	Blocked 5 hours
Regular Member	Local Resource Table	Data scraping	Blocked 2 days

The JC will have the following ABIs for judging misbehavior:

• *misbehaviorJudge()*: This ABI will be run by any ACC to report the misbehavior of a subject to the JC. After receiving the report, this ABI judges the misbehavior of the subject, determines the penalty on the subject and returns the penalty decision to the ACC. This ABI also adds a new misbehavior record to the list.

3.2.4 Register contract (RC)

The RC will maintain a lookup table, which registers the information to find and execute all the methods. An example:

• *MethodName*: the name of the method;

• *Subject*: the subject of the corresponding subject-object pair of the method;

• *Object*: the object of the corresponding subject-object pair of the method;

• *ScName*: the name of the corresponding smart contract implementing this method;

• *Creator*: the peer who created and deployed the contract;

• *ScAddress*: the address of the smart contract; and

MethodName	**Subject**	**Object**	**ScName**	**ScAddress**	**ABI**
viewGlobalResource	Primary Group Head	Global Resource Table	Primary Group Head Role ACC	0x123...def	AccessControl()
viewLocalResource	Primary Group Head	Local Resource Table	Primary Group Head Role ACC	0x123...def	AccessControl()
communicateWithOwnGroupHead	Regular Member	Primary Group Head	Regular Members Role ACC	0x456...ghi	AccessControl()
viewGlobalResource	Secondary Group Head	Global Resource Table	Secondary Group Head Role ACC	0x789...jkl	AccessControl()

With the help of the lookup table, the RC provides the following main ABIs to mange these methods.

• *methodRegister()*: This ABI receives the information of a new method and registers the information into the lookup table.

• *methodUpdate()*: This ABI receives the information of an existing method that needs to be updated and up-date the information, especially the fields of ScAddress and ABI.

• *methodDelete()*: This ABI receives the *MethodName* of a method and deletes the method from the lookuptable.

• *getContract()*: This ABI receives the *MethodName* of a method and returns the address and ABIs of the contract (i.e., the ACCs and JC) of the method.

3.2.5 Threat Model

We also adopted a threat model that studies the vulnerabilities of the group heads that relate to the challenges for data integrity. It assumes a scenario where compromise to a primary group head takes place due to malicious activities, in which case a secondary group head will be maintaining the integrity of inter-group communications.

3.2.6 Communication Simulation

3.2.6.1 Dynamic Reference Passing

Beyond maintaining the lookup table, the RC is also engineered to dynamically provide reference data to different roles within the network based on their needs. For instance:

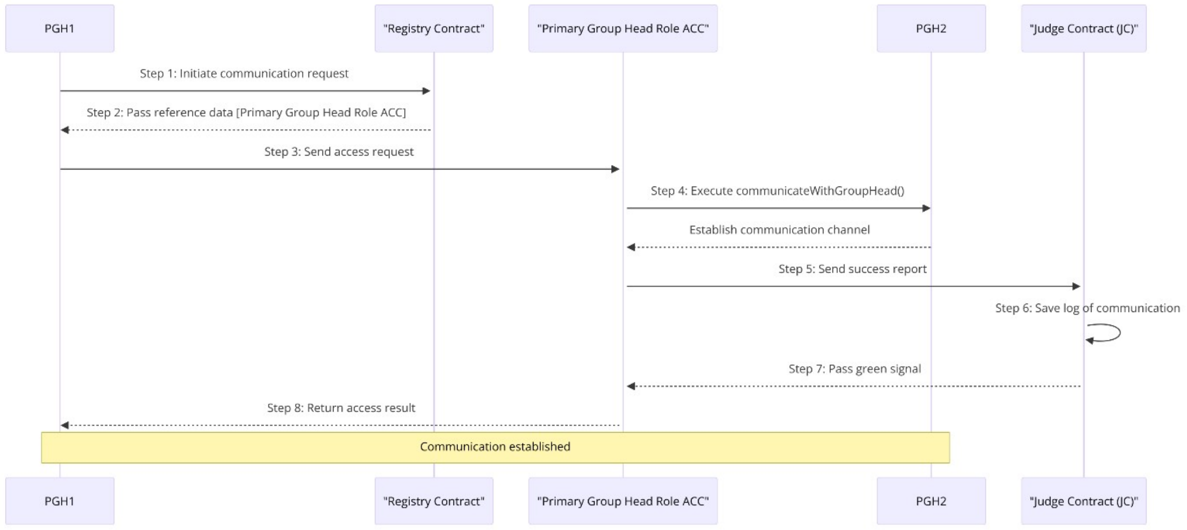
- When a **Primary Group Head** acts as the subject and requests access to a resource or interaction, the RC supplies the reference data, such as the "Primary Group Head Role ACC," to facilitate the access or communication.
- Similarly, if a **Secondary Group Head** or **Regular Member** assumes the role of the subject, intending to connect with a resource or another entity, the RC provides them with their respective ACC references, like "Secondary Group Head Role ACC" or "Regular Members Role ACC," accordingly.

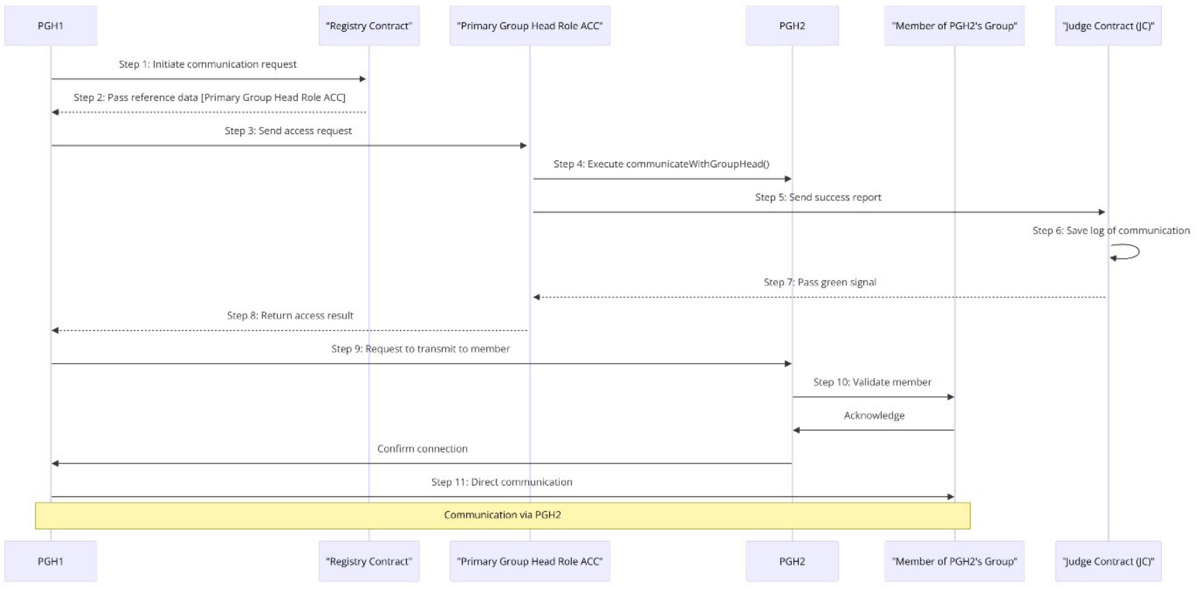
Subject Role	**Reference Data Supplied by RC**
Primary Group Head	Primary Group Head Role ACC
Secondary Group Head	Secondary Group Head Role ACC
Regular Member	Regular Member Role ACC

This dynamic reference mechanism ensures that each peer, depending on their role and the nature of the access request, is equipped with the appropriate smart contract references. It facilitates a fluid and secure interaction within the network, ensuring that access controls are adhered to, and that the P2P ecosystem operates within the bounds of predefined policies and permissions. Through this sophisticated orchestration, the RC significantly enhances the network's security, reliability, and efficiency.

3.2.6.2 Workflow Diagram

In our blockchain-based system, the communication between primary group heads (PGHs) is efficiently managed through a series of interactions with smart contracts, specifically the Registry Contract and the Primary Group Head Role Access Control Contract (ACC). These interactions ensure secure and verified communications within the network. Detailed steps of the communication process are fully illustrated in the accompanying diagrams, which depict how access requests are handled, validated, and logged to maintain network security and integrity.


PGH1
"Registry Contract"
"Primary Group Head Role ACC"
PGH2
"Judge Contract (JC)"
Step 1: Initiate communication request
Step 2: Pass reference data [Primary Group Head Role ACC]
Step 3: Send access request
Step 4: Execute communicateWithGroupHead()
Establish communication channel
Step 5: Send success report
Step 6: Save log of communication
Step 7: Pass green signal
Step 8: Return access result
Communication established


PGH1
"Registry Contract"
"Primary Group Head Role ACC"
PGH2
"Member of PGH2's Group"
"Judge Contract (JC)"
Step 1: Initiate communication request
Step 2: Pass reference data [Primary Group Head Role ACC]
Step 3: Send access request
Step 4: Execute communicateWithGroupHead()
Step 5: Send success report
Step 6: Save log of communication
Step 7: Pass green signal
Step 8: Return access result
Step 9: Request to transmit to member
Step 10: Validate member
Acknowledge
Confirm connection
Step 11: Direct communication
Communication via PGH2

In the blockchain-based access control system, the architecture is meticulously structured to ensure that updates and deletions of Access Control Contracts (ACCs) or their methods are strictly regulated. This structured approach ensures that any changes made are in line with the overarching system's integrity and security standards.

Operational guidelines are put in place to manage these updates effectively. Each change to access control methods or policies requires a thorough review process to confirm its alignment with the system's predefined roles and architecture. This ensures that modifications do not compromise the network’s security or functionality.

Both the Register Contract (RC) and Judge Contract (JC) are instrumental in maintaining the integrity of the system. They oversee the application of policies and the execution of roles within the network, particularly ensuring that primary group heads (PGHs) can interact seamlessly and securely. This oversight helps tailor access control mechanisms to meet the specific needs of the network’s architecture, enhancing both communication and operational efficiency across the platform.

3.3 Smart Contract Implementation

3.3.1 Specifications of Devices

This section provides in implementation to demonstrate the application of the proposed framework for distributed access control. We first introduce the hardware and software used in the study and then present how the access control is implemented based on the framework.

Hardware used in this implementation:

Device	**CPU**	**Operating System**	**Memory**	**Storage**
HP Probook G4 440	Processor Intel(R) Core(TM) i7-7500U CPU @ 2.70GHz, 2904 Mhz, 2 Core(s), 4 Logical Processor(s)	Microsoft Windows 11 Pro (64 bit)	15.9 GB	HDD: 931.51 GB SSD: 223.57 GB

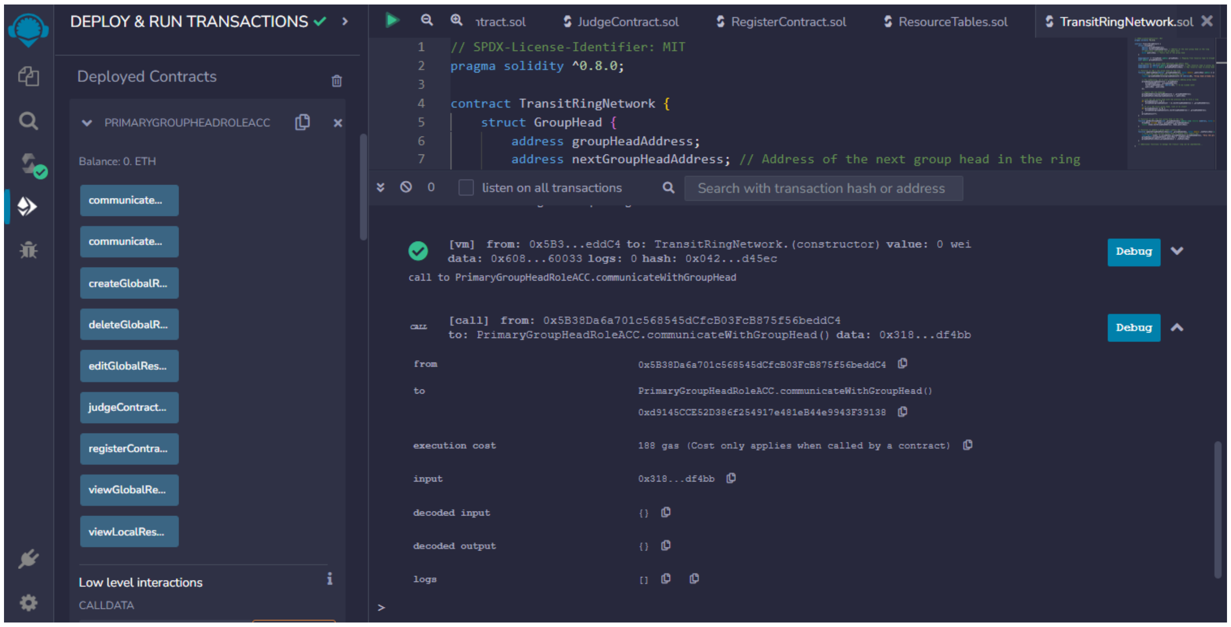
3.3.2 System Implementation

1. **Access Control Contracts (AccessControlContracts.sol)**: Defines roles and permissions for different group members, implementing functions for resource access and management.

```
Algorithm AccessControl
NormalPrimaryGH(resourceTable){
if resourceTable == ("Global Resource Table", "Local Resource Table):
return “Full access: view, edit, create, delete”
NormalSecondaryGH(resourceTable){
if resourceTable == ("Global Resource Table", "Local Resource Table) :
return view
```

```
NormalRegularMember(resourceTable){
if resourceTable == ("LocalResource Table"):
return view
else resourceTable == ("Global Resource Table"):
return deny
End Algorithm
```

Figure: In figure 3, we observe the execution of a contract named "PrimaryGroupHeadRoleACC." This contract is tasked with managing communications between group heads. It demonstrates a transaction initiated from a specific Ethereum address, signaling an attempt to invoke the method communicateWithGroupHead. The transaction incurs a gas cost, which is the fee for the computational effort needed to execute the operation—denoted as 188 gas units in this case. Gas costs are only applicable when functions are called by a contract. The screenshot also shows the input data along with decoded input, output, and logs, which collectively confirm the successful execution of the contract's function.



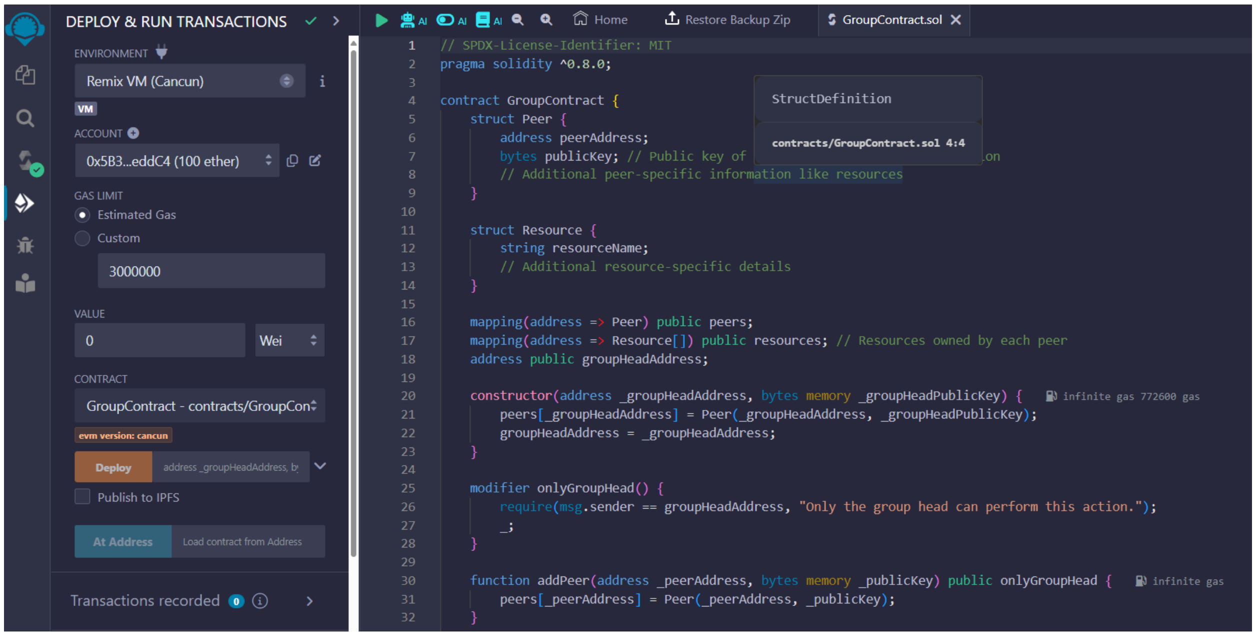
```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

contract TransitRingNetwork {
    struct GroupHead {
        address groupHeadAddress;
        address nextGroupHeadAddress; // Address of the next group head in the ring
```

```
[vm] from: 0x5B3...eddC4 to: TransitRingNetwork.(constructor) value: 0 wei
data: 0x608...60033 logs: 0 hash: 0x042...d45ec
call to PrimaryGroupHeadRoleACC.communicateWithGroupHead

CALL [call] from: 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4
to: PrimaryGroupHeadRoleACC.communicateWithGroupHead() data: 0x318...df4bb

from              0x5B38Da6a701c568545dCfcB03FcB875f56beddC4
to                PrimaryGroupHeadRoleACC.communicateWithGroupHead()
                  0xd9145CCE52D386f254917e481eB44e9943F39138
execution cost    188 gas (Cost only applies when called by a contract)
input             0x318...df4bb
decoded input     {}
decoded output    {}
logs              []
>
```

2. **Group Management Contracts (GroupContracts.sol)**: Manages group members and resources, with functions for adding peers and assigning resources.

```
Algorithm GroupContract
- Initialize group with a head and public key
- addPeer(peerAddress, publicKey):
  IF called by group head THEN
    Add peer to group
  ENDIF
- addResource(peerAddress, resourceName):
  IF called by group head THEN
    Assign resource to peer
  ENDIF
End Algorithm
```

Figure: Implementation illustrated in Figure 4.



ENVIRONMENT: Remix VM (Cancun)
ACCOUNT: 0x5B3...eddC4 (100 ether)
GAS LIMIT: Estimated Gas / Custom 3000000
VALUE: 0 Wei
CONTRACT: GroupContract - contracts/GroupCon
evm version: cancun

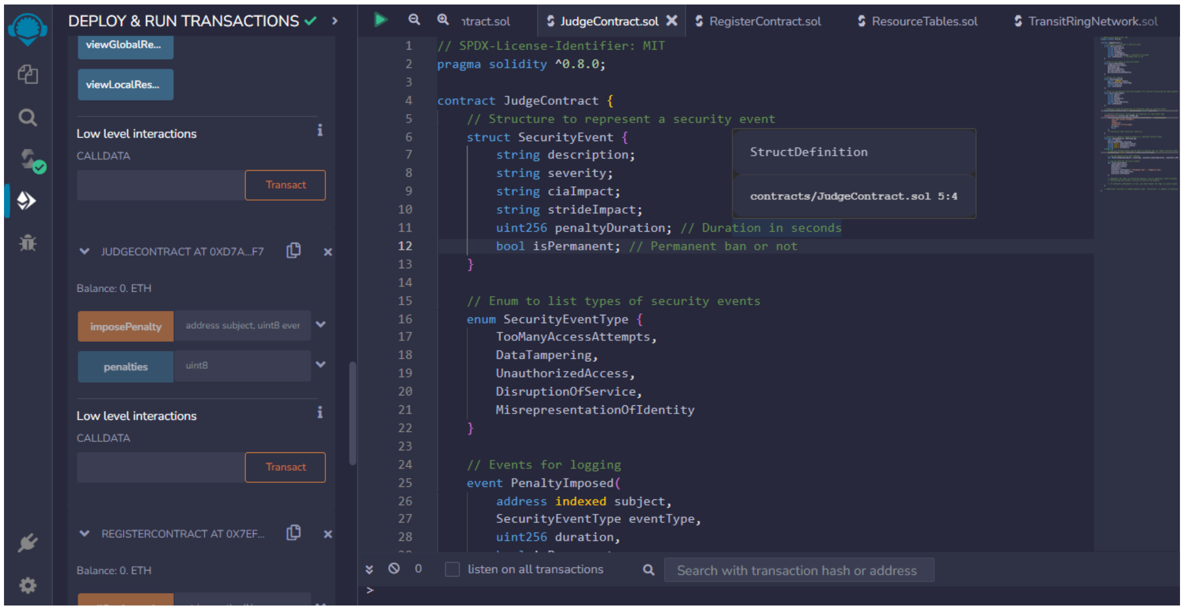
```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

contract GroupContract {
    struct Peer {
        address peerAddress;
        bytes publicKey; // Public key of                              on
        // Additional peer-specific information like resources
    }

    struct Resource {
        string resourceName;
        // Additional resource-specific details
    }

    mapping(address => Peer) public peers;
    mapping(address => Resource[]) public resources; // Resources owned by each peer
    address public groupHeadAddress;

    constructor(address _groupHeadAddress, bytes memory _groupHeadPublicKey) {    infinite gas 772600 gas
        peers[_groupHeadAddress] = Peer(_groupHeadAddress, _groupHeadPublicKey);
        groupHeadAddress = _groupHeadAddress;
    }

    modifier onlyGroupHead() {
        require(msg.sender == groupHeadAddress, "Only the group head can perform this action.");
        _;
    }

    function addPeer(address _peerAddress, bytes memory _publicKey) public onlyGroupHead {    infinite gas
        peers[_peerAddress] = Peer(_peerAddress, _publicKey);
    }
```

3. **Judicial System Contracts (JudgeContracts.sol)**: Handles security events and imposes penalties based on misbehaviors, crucial for enforcing the network's access control policies.

```
Algorithm JudgeContract
- Define security events and corresponding penalties
- imposePenalty(subject, eventType):
  IF eventType detected THEN
    Log event and apply penalty (temporary or permanent)
  ENDIF
End Algorithm
```

Figure: In the figure 5, we have "JudgeContract," which empowers the network with the ability to impose penalties. The left panel displays an interface for imposing penalties, where a specific address, identified as the subject of the penalty, can be targeted. The penalties are defined within the contract and can be executed through its ABI, reflecting the Judge

Contract's crucial role in maintaining order and enforcing rules within the network.



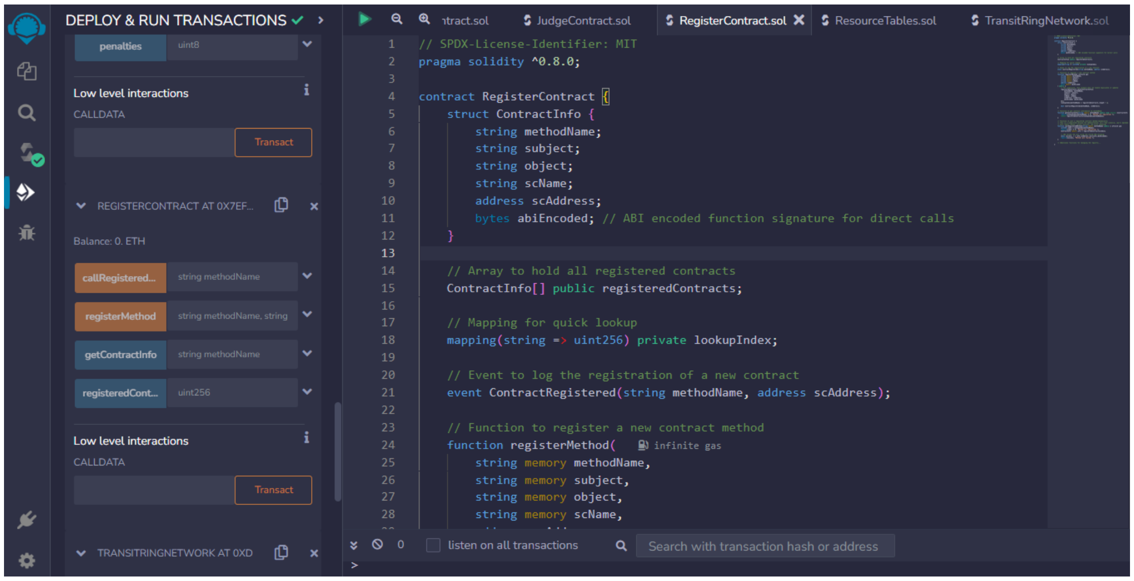
```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

contract JudgeContract {
    // Structure to represent a security event
    struct SecurityEvent {
        string description;
        string severity;
        string ciaImpact;
        string strideImpact;
        uint256 penaltyDuration; // Duration in seconds
        bool isPermanent; // Permanent ban or not
    }

    // Enum to list types of security events
    enum SecurityEventType {
        TooManyAccessAttempts,
        DataTampering,
        UnauthorizedAccess,
        DisruptionOfService,
        MisrepresentationOfIdentity
    }

    // Events for logging
    event PenaltyImposed(
        address indexed subject,
        SecurityEventType eventType,
        uint256 duration,
```

Infrastructure Support Contracts:

4. **Registry System (RegisterContracts.sol)**: Maintains a comprehensive registry of all contracts, enhancing the interaction and lookup processes within the network.

```
Algorithm RegisterContract
- registerMethod(methodName, details):
  Add contract method details to registry
- getContractInfo(methodName):
  Return contract method details from registry
- callRegisteredMethod(methodName):
  Dynamically call a method from the registered contract details
End Algorithm
```

Figure: "RegisterContract," illustrated in the figure 5, which maintains the registry of all methods and associated contracts within the network. With functionalities such as registerMethod, getMethod, and registerContractInfo, the Register Contract serves as a directory service, enabling dynamic linking between contracts and methods. It ensures that interactions between different network entities are conducted seamlessly and securely.



```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

contract RegisterContract {
    struct ContractInfo {
        string methodName;
        string subject;
        string object;
        string scName;
        address scAddress;
        bytes abiEncoded; // ABI encoded function signature for direct calls
    }

    // Array to hold all registered contracts
    ContractInfo[] public registeredContracts;

    // Mapping for quick lookup
    mapping(string => uint256) private lookupIndex;

    // Event to log the registration of a new contract
    event ContractRegistered(string methodName, address scAddress);

    // Function to register a new contract method
    function registerMethod(
        string memory methodName,
        string memory subject,
        string memory object,
        string memory scName,
```

5. **Network Structure and Resource Management:**

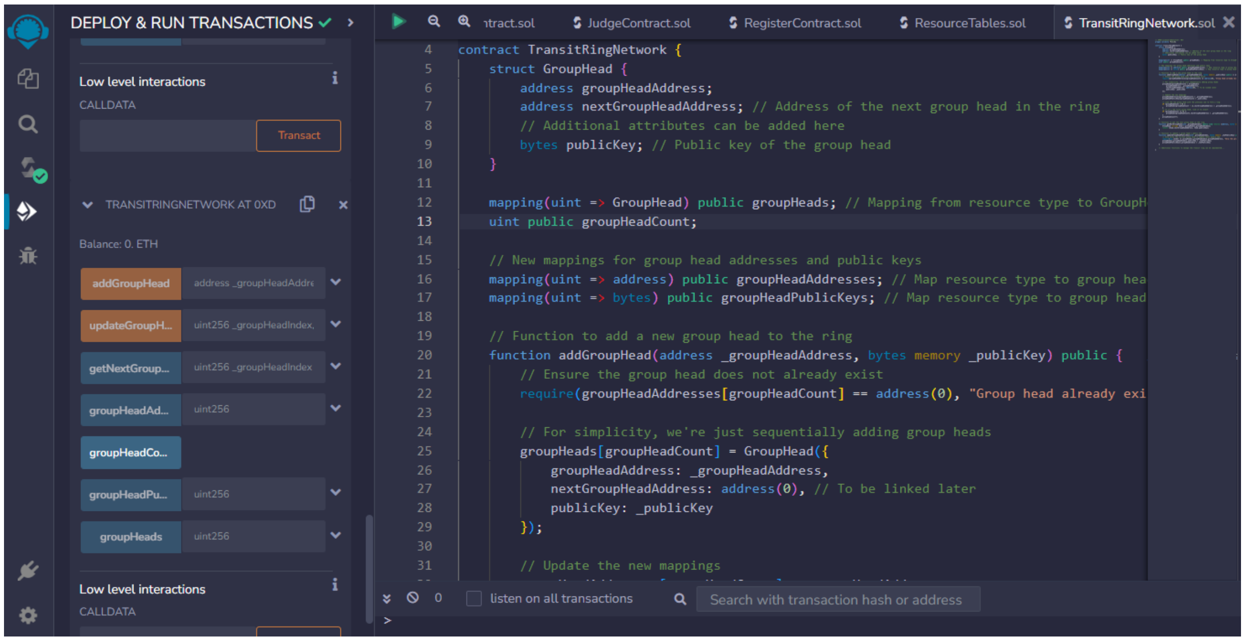
- **Resource Table (ResourceTable.sol)**: Manages a local resource table for each peer, crucial for resource allocation within groups.
- **Transit Ring Network (TransitRingNetwork.sol)**: Facilitates efficient data lookup and inter-group communication through a managed ring network of group heads.

Algorithms: **ResourceTable** and **TransitRingNetwork**

```
Algorithm ResourceTable
- Initialize group with a head
- addResource(peerAddress, resourceName):
  IF called by group head THEN
    Add resource to peer's local resource table
  ENDIF
End Algorithm
```

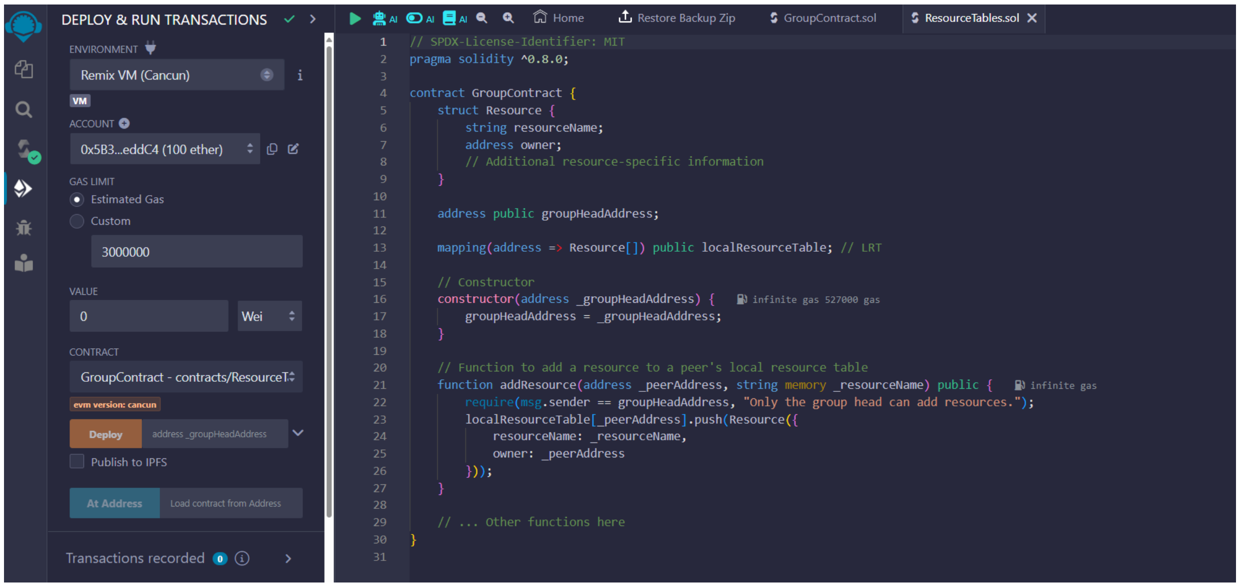
```
Algorithm TransitRingNetwork
- Initialize empty ring network of group heads
- addGroupHead(groupHeadAddress, publicKey):
  Add new group head to the ring, link to next
- getNextGroupHead(groupHeadIndex):
  Return next group head in the ring
-updateGroupHeadPublicKey(groupHeadIndex, newPublicKey):
  Update group head's public key if called by the group head itself
End Algorithm
```

Figure: "TransitRingNetwork," which lays the foundation of our P2P architecture. This contract allows for the addition and updating of group head information, facilitating the expansion or alteration of the network's structure. While the current architecture is set, these functionalities hint at future potential for network growth or reconfiguration.



```
contract TransitRingNetwork {
    struct GroupHead {
        address groupHeadAddress;
        address nextGroupHeadAddress; // Address of the next group head in the ring
        // Additional attributes can be added here
        bytes publicKey; // Public key of the group head
    }

    mapping(uint => GroupHead) public groupHeads; // Mapping from resource type to GroupH
    uint public groupHeadCount;

    // New mappings for group head addresses and public keys
    mapping(uint => address) public groupHeadAddresses; // Map resource type to group hea
    mapping(uint => bytes) public groupHeadPublicKeys; // Map resource type to group head

    // Function to add a new group head to the ring
    function addGroupHead(address _groupHeadAddress, bytes memory _publicKey) public {
        // Ensure the group head does not already exist
        require(groupHeadAddresses[groupHeadCount] == address(0), "Group head already exi

        // For simplicity, we're just sequentially adding group heads
        groupHeads[groupHeadCount] = GroupHead({
            groupHeadAddress: _groupHeadAddress,
            nextGroupHeadAddress: address(0), // To be linked later
            publicKey: _publicKey
        });

        // Update the new mappings
```

Figure: "ResourceTable," allows deploying the group head address, facilitating information sharing within of the permitted network's structure. While the current architecture is set, these functionalities hint at future potential for network growth or reconfiguration.



ENVIRONMENT: Remix VM (Cancun)
ACCOUNT: 0x5B3...eddC4 (100 ether)
GAS LIMIT: Estimated Gas [✓] / Custom: 3000000
VALUE: 0 Wei
CONTRACT: GroupContract - contracts/ResourceT
evm version: cancun
Deploy: address _groupHeadAddress

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

contract GroupContract {
    struct Resource {
        string resourceName;
        address owner;
        // Additional resource-specific information
    }

    address public groupHeadAddress;

    mapping(address => Resource[]) public localResourceTable; // LRT

    // Constructor
    constructor(address _groupHeadAddress) {      infinite gas 527000 gas
        groupHeadAddress = _groupHeadAddress;
    }

    // Function to add a resource to a peer's local resource table
    function addResource(address _peerAddress, string memory _resourceName) public {    infinite gas
        require(msg.sender == groupHeadAddress, "Only the group head can add resources.");
        localResourceTable[_peerAddress].push(Resource({
            resourceName: _resourceName,
            owner: _peerAddress
        }));
    }

    // ... Other functions here
}
```

In this implementation, the Ethereum-based Remix Integrated Development Environment (IDE) was utilized for the development of smart contracts that underpin the access control mechanisms within our peer-to-peer network. The Remix IDE enabled the efficient writing, testing, and deployment of smart contracts written in Solidity directly onto the Ethereum blockchain. The development and testing phases of these contracts were carried out within the Remix IDE, which provided an Ethereum node simulation for deploying and invoking the smart contracts. This allowed for a thorough examination of the contracts' functionality before being deployed to the live network.

This Ethereum-based implementation leverages the security and transparency of blockchain technology while allowing for the complex, rule-based access control logic necessary for secure its applications. The combination of Ethereum's robust blockchain with the flexible programming afforded by Remix and Solidity provides a powerful toolset for realizing distributed access control systems in the IoT domain.

3.4 Future Plan

This thesis aims to explore innovative blockchain applications, evaluate their effectiveness in real-world scenarios, and push the boundaries of current access control technologies.

1. Theoretical Simulation

Minor Goals:

- **(Event 1 – Normal Operation)** *Without Attack*: Simulate the event where and check the framework's baseline security in a controlled environment.
- **(Event 2 – Normal Operation)** *With Attack but No Access Control:* Simulate the network's vulnerability to security breaches and the effects of such breaches on network.
- **(Event 3 – Malicious Operation)** *With Attack but Working Access Control*: Evaluate the framework's effectiveness in real-time misbehavior detection (ACC’s), Impose Penalty (JC) and storing the log data of an attack and helping the JC and ACC’s to safeguard their responsibility (RC) in the process of mitigating an attack.

2. Engineered Solution:

Minor Goals:

- **Smart Contract for Onboarding**: Create a smart contract that manages the registration and validation of new peers. This contract will require new members to submit cryptographic proof of identity and credentials.
- **Validation Protocol:** Implement a protocol within the smart contract that allows existing network nodes to vote on the admission of new peers based on the submitted credentials and a consensus algorithm.
- **Credential Verification:** Integrate a cryptographic verification mechanism that checks the validity of the credentials against a decentralized public key infrastructure (PKI).

3. Integration of Advanced Blockchain Features

Minor Goals:

- Explore the integration of additional blockchain functionalities, such as zero-knowledge proofs and sidechains.
- Develop and test smart contract upgrades that introduce more complex rule sets for dynamic access control.

4. Expansion of the Access Control Framework

Minor Goals:

- Design and deploy additional roles and permissions within the network to contain more complex structures.
- Modifying access control contracts to automatically respond to common security incidents, in this manner reducing the need for manual intervention each time.

5. Future Directions

Minor Goals:

- Explore the use of artificial intelligence (AI) to predict security breaches before they occur, through integrating AI with blockchain.

- Prove the scalable and effective nature of the framework with a real-world IoT environment pilot, e.g., in a smart city.

References

1. O. Novo, "Blockchain Meets IoT: An Architecture for Scalable Access Management in IoT," IEEE Internet of Things Journal, vol. 5, no. 2, pp. 1184-1195, Apr. 2018.
2. Y. Sun, H. Song, Z. Tu, and Y. Qin, "Blockchain for IoT Access Control: Recent Trends and Future Research Directions," Sensors (Basel, Switzerland), 2022.
3. H. Yu, T. Chen, and J. Wang, "A blockchain-based Access Control Mechanism for IOT," in Proceedings of the 2022 6th International Conference on Electronic Information Technology and Computer Engineering, 2022.
4. M. A. Islam and S. Madria, "A Permissioned Blockchain Based Access Control System for IOT," in 2019 IEEE International Conference on Blockchain (Blockchain), 2019.
5. S. Sun, S. Chen, R. Du, W. Li, and D. Qi, "Blockchain Based Fine-Grained and Scalable Access Control for IoT Security and Privacy," in 2019 IEEE Fourth International Conference on Data Science in Cyberspace (DSC), 2019.
6. Riabi, Y. Dhif, H. K. B. Ayed, and K. Zaatouri, "A Blockchain based access control for IoT," in 2019 15th International Wireless Communications & Mobile Computing Conference (IWCMC), 2019.
7. Y. Zhang, A. Memariani, and N. Bidikar, "A Review on Blockchain-based Access Control Models in IoT Applications," in 2020 IEEE 16th International Conference on Control & Automation (ICCA), 2020.
8. H. Liu, D. Han, and D. Li, "Fabric-iot: A Blockchain-Based Access Control System in IoT," IEEE Access, 2020.
9. D. Han, Y. Zhu, D. Li, W. Liang, A. Souri, and K.-C. Li, "A Blockchain-Based Auditable Access Control System for Private Data in Service-Centric IoT Environments," IEEE Transactions on Industrial Informatics, 2022.
10. Z. Li, J. Hao, J. Liu, H. Wang, and M. Xian,"An IoT-Applicable Access Control Model Under Double-Layer Blockchain," IEEE Transactions on Circuits and Systems II: Express Briefs, 2021.
11. Y. Liu, Q. Lu, S. Chen, Q. Qu, H. O'Connor, K. Choo, and H. Zhang, "Capability-based IoT access control using blockchain," Digit. Commun. Networks, 2020.
12. S. Mbarek, M. Ge, and T. Pitner, "Blockchain-Based Access Control for IoT in Smart Home Systems," 2020.

13. T. Sultana, A. S. Almogren, M. Akbar, M. Zuair, I. Ullah, N. Javaid, "Data Sharing System Integrating Access Control Mechanism using Blockchain-Based Smart Contracts for IoT Devices," Applied Sciences, 2020.
14. "Access Control Mechanism for IoT using Blockchain," International Journal of Recent Technology and Engineering, 2020.
15. G. Badhe and D. M. Arjunwadkar, "Access Control Systems Based on Blockchain Technology," international journal of engineering technology and management sciences, 2023.
16. N. Ding, J. Cao, C. Li, K. Fan, and H. Li, "A Novel Attribute-Based Access Control Scheme Using Blockchain for IoT," IEEE Access, 2019.
17. S. Algarni, F. Eassa, K. Almarhabi, A. Almalaise, E. Albassam, K. Alsubhi, and M. Yamin, "Blockchain-Based Secured Access Control in an IoT System," Applied Sciences, 2021.
18. B. Bera, S. Saha, S. Das, and A. Vasilakos, "Designing Blockchain-Based Access Control Protocol in IoT-Enabled Smart-Grid System," IEEE Internet of Things Journal, 2021.
19. Y. Wang, N. Xu, H. Zhang, W. Sun, and A. Benslimane, "Dynamic Access Control and Trust Management for Blockchain-Empowered IoT," IEEE Internet of Things Journal, 2022.
20. S. Badhe, G. Arjunwadkar, "Access Control Systems Based on Blockchain Technology," International Journal of Engineering Technology and Management Sciences, 2023.